Assessing Site-Geometry for Architectural Design Using Graph Theory

Horvath, Anca-Simona

Published in:
Proceedings of the Second International Conference for PhD students in Civil Engineering and Architecture

DOI (link to publication from Publisher):
10.6084/m9.figshare.14555541.v1

Publication date:
2014

Document Version
Publisher's PDF, also known as Version of record

2nd International Conference for PhD students in Civil Engineering and Architecture
CE-PhD 2014, 10-13 December 2014, Cluj-Napoca, Romania
www.cephd.ro

Assessing Site-Geometry for Architectural Design Using Graph Theory

Anca S. Horvath*[1]

[1] *Technical University of Cluj-Napoca, Faculty of Architecture and Urban Planning, 72-76 Observatorului Str., 400489, Cluj-Napoca, Romania*

Abstract

This paper presents six node components from Tardigrade, an add-on developed for the visual programming language Grasshopper which works with Rhinoceros 3D. Tardigrade is a work in progress and will implement various general use nodes (/libraries/subroutines) that enhance Grasshopper's functionality. The components presented here employ four classic graph-search algorithms - Breadth First Search, Greedy Best Search, Dijkstra and A. Widely used in computer science, graph theory was developed based on real-world urban problems but is now seldom used by architects. The six components are general use, but were originally meant for topography analysis as decision making helpers based on Space Syntax. Often, an entirely empty, relatively big plot is offered for refurbishment. Lacking relevant landmarks, it is difficult to decide on road and plot positioning. Graph theory can be used to establish road configurations by computing cost-based paths between a start and an end-point or between a start and all other graph points. Site topography can come as meshes, free-form surfaces, level curves or 3D points. Any of these is easily reduced to a finite list of points, each with its x, y, z coordinates. The components work for 3D point collections regardless of their data structure. The graph's edges are reconstructed dynamically from the node's neighbors. They offer an otherwise lacking, easy topography analysis tool for Grasshopper users.*

Keywords: graph theory, architecture, Grasshopper, Rhino 3D, cost-effective path.

1. Introduction

The use of computation for solving complex design problems in the field of architecture has been growing timidly over the last two decades. This is due to the convergence of a general increased interest in the digital's role in retooling the profession coupled with the proliferation of more intuitive, easy to learn and read, designer-centered programming languages. Unsurprisingly, visual programming, where direct interaction with text-code is minimal and snapping together blocks of logic is instinctive, quickly became one of the more popular choices among the visually savvy architects. Grasshopper 3D, developed by David Rutten at McNeel is a visual programming language which runs with the NURBS modeling software Rhino 3D. Grasshopper has an advanced GUI. Node components with various functions are placed on the canvas and data is passed via wires from node to node. A study in 2014 analyzed 928 jobs at the top 50 architectural firms in the world and compiled the software requirements listed for each job. Grasshopper, the only programming language on the list, was a prerequisite in 3% of the job offers, just under hand sketching (4%). This

* Corresponding author: Tel./ Fax.: 0744 79 17 17
E-mail address: ancas.horvath@gmail.com

ranking is way below Revit (71%), Autocad (50%), Sketchup (34%), 3D Max (28%) or Rhino 3D (10%), but above the more popular Romanian BIM choices of Archicad - 0% and Nemetscheck which is not on the list at all [1]. However, the study looked at the firms with the highest turnaround, not those considered to have the biggest impact for the profession. "Starchitect" practices like ZHA, Norman Foster or OMA, known to employ unconventional software were ignored. In reality, there is a high chance programming is even more in demand at the forefront of architecture. Grasshopper (introduced in 2007) is not the only dedicated visual programming language to extend initial capabilities for modeling software. In 2012 Sverchok for Blender and DesignScript for Autocad (which became Dynamo just days ago) were released; in 2013, Dynamo for Revit - [2] and [3]. On the other hand, text based scripting environments are offered in most CAD packages and scripting node components exist for all the dedicated programming languages described above. Python, with its simple syntax, is available in many architecturally relevant software packages: Rhino 3D supports Python along Rhinoscript [4]; Grasshopper has a Python Script component (in addition to its more seasoned VB.net and C#) [5]; Revit and its open source counterpart Vasari support Iron Python (although accessing the Revit API is "complex and no serious scripting culture is available around BIM tools in general" [4]); Dynamo only offers custom node creation using Python; Maya supports Python along its MELscript; Blender supports Python in its scripting mode IDE; Sverchok is written in Python altogether; 3D Max has a Python Script extension. This inventory will probably grow larger in time with Python's soaring popularity in general. With the ever-rising plethora of software applications intended for and used in architecture, Python might even become a meta-narrative.

2. Motivation

Using visual programming languages for advanced architectural design offers more geometrical freedom and the ability to compute solutions to complex problems. Despite the steep learning curve and accessibility of these languages, one setback is that designers without prior exposure to programming tend to remain uneducated in the field of computation even well after engaging in parametric modeling, as has been shown or suggested in [5], [6], [7] and [8]. Thus, in spite of graph theory's historical link to urban understanding, this classical programming concept is not-so-often used by architects. A reason or a consequence of this is the lack of direct, simple tools to assess topography in most of the software packages aforementioned.

Space Syntax have developed models for urban design and architecture based on graph theory. Their trial is to create a new kind of knowledge which helps architects decide on how well their designs might perform, what their solutions mean and imply [9]. DepthmapX is "a multi-platform software application that performs a set of spatial network analyses designed to understand social processes within the built environment. DepthmapX is developed by Tasos Varoudis at UCL's Space group" [10]. The stand-alone depthmapX offers advanced investigations, from graph theory implementation to agent-based modeling, but, albeit a great tool, it is a lot less used (than popular software applications and the visual programming languages that come with them); it is also 2-dimensional. The SpiderWeb add-on for Grasshopper was developed to enable "various generative approaches based on Space Syntax. It provides the basics to copy some of the analysis available in dephmapX" [11]. Spider-Web, introduced in 2011, is written in VB.net and includes a comprehensive set of graph related components which allow many uses. However, it specifically employs graphs and remains complex even for more advanced users. One other add-in for Grasshopper, namely Shortest Walk developed by Antonio Turiello exposes one single component which implements a topology calculator based on the A* logic. Shortest Walk was developed for Grasshopper 0.800009, an update was made for version 0.90014 (in 2011) but it does not work with Rhino 5 nor the current version of Grasshopper 1 [12]. The six components presented in this paper

are less complex than dephmapX and overlap some of SpiderWeb's own nodes or clusters. However, they provide further, more detailed functionality and hopefully, easier interaction for users less knowledgeable of graph theory. Additionally, in light of the arguments presented in the introduction, the nodes are written in Python with the aim to easily port the code into other visual programming languages or scripting IDE's. The code was partly influenced by Amit Patel's blog on game design [13].The upcoming goals are PythonScript for Rhino and Dynamo for Revit.

3. Tardigrade path finding

Figure 1. - Tardigrade menu

Figure 1 shows Tardigrade's path finding menu inside Grasshopper, within Rhino 5.0.

3.1 Input

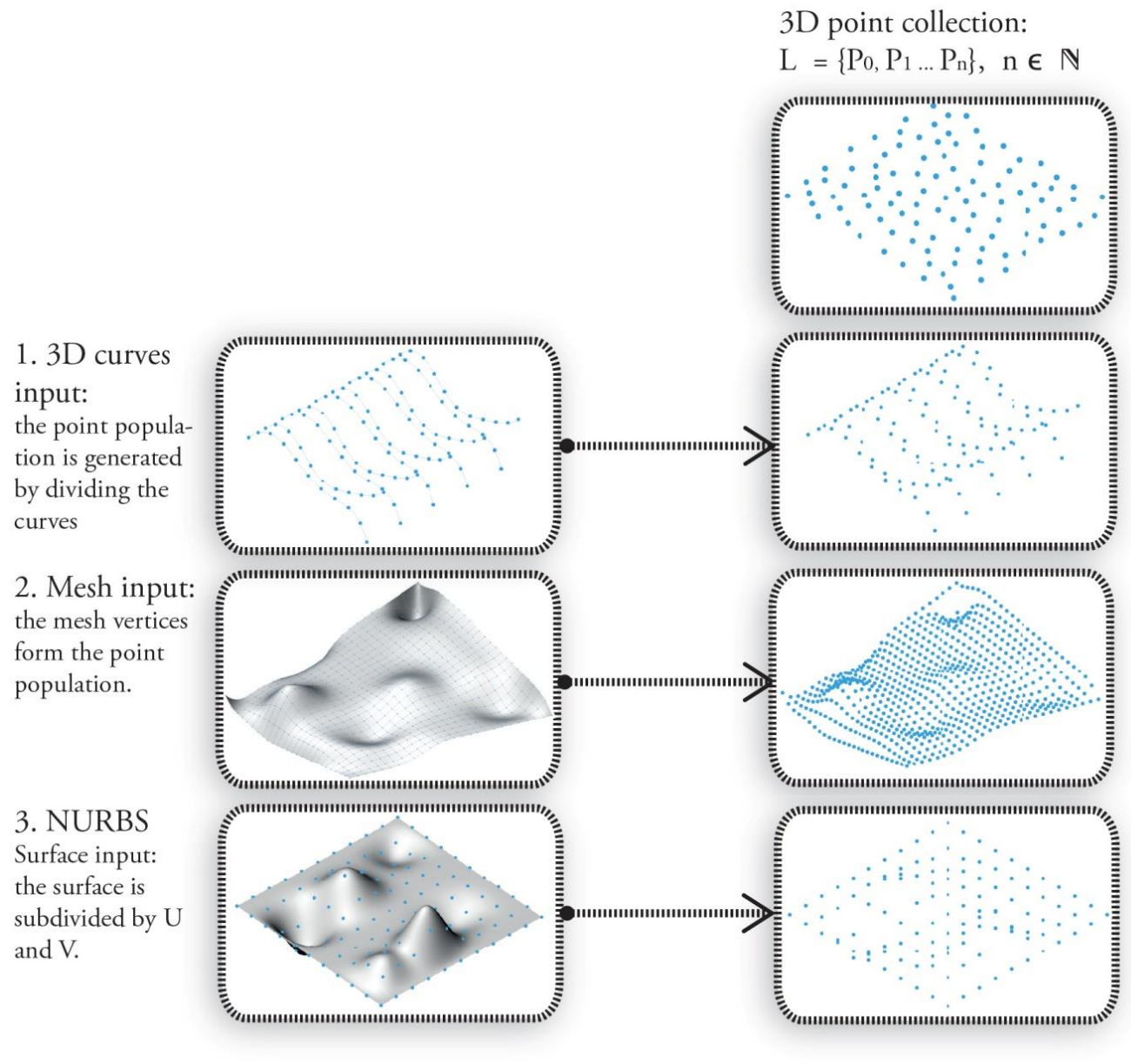

Figure 2. Point cloud input for the 6 Tardigrade components

The six components take a 3D point collection as input and transform it into a graph. The characteristics of the collection are visually described in Figure 2 and mathematically described as follows:

$$G\{P,U\},U=\emptyset \quad (1)$$
$$P = \{P_0,P_1 \ldots P_n\},\ n \in \mathbb{N} \quad (2)$$
$$Pi(Xi, Yi, Zi),\ i \in \{0,n\} \quad (3)$$
$$Xi, Yi, Zi \in \mathbb{R} \quad (4)$$
$$X_{max} = \max\{X_i\} \quad (5)$$
$$X_{min} = \min\{X_i\} \quad (6)$$
$$Y_{max} = \max\{Y_i\} \quad (7)$$
$$Y_{min} = \min\{Y_i\} \quad (8)$$
$$P(X_{min}, Y_{min}, Z_i) \in P \quad (9)$$
$$P(X_{max}, Y_{min}, Z_i) \in P \quad (10)$$
$$P(X_{min}, Y_{max}, Z_i) \in P \quad (11)$$
$$P(X_{max}, Y_{max}, Z_i) \in P \quad (12)$$
$$A(X_a, Y_a, Z_a) \in P \quad (13)$$
$$B(X_b, Y_b, Z_b) \in P \quad (14)$$

Restrictions (9) - (12) are based on the way graph edges (U) are reconstructed. The method employed internalizes the k input from the k-NN algorithm.

3.2. Edge reconstruction - nearest neighbor search problem

In the widely used k-NN search, the nearest k neighbors to a vertex A are the first k elements in the vertex list ordered by Cartesian distance to A [16]. The six Tardigrade components take a null graph and reconstruct the edges dynamically at each step. They use the k-NN algorithm, but Tardigrade reduces input and definition complexity. No minimum distance between nodes for graph reconstruction (fixed-radius near neighbors) nor the k number of neighbors is required(k-NN), it is solved internally. This works for a relatively rectangular grid, where the number of neighbors can be evaluated (almost naively) according to the node's x,y position as illustrated in Figure 4. The node assessment, albeit made easier for the user, and more efficient for a rectangular grid, causes problems in cases where the grid ratio is $n*u<\sqrt{u^2 + t^2}$, $n>1$ (see Figure 3). Spiderweb's "Reconstruct graph from points" node works with a minimum distance (fixed-radius nearest neighbors) creating edges between nodes based on a distance provided by the user. Further research is needed to establish if calculating correct neighbors for edge reconstruction for any grid form without prior user input of distance or neighbor number is possible, feasible and relevant.

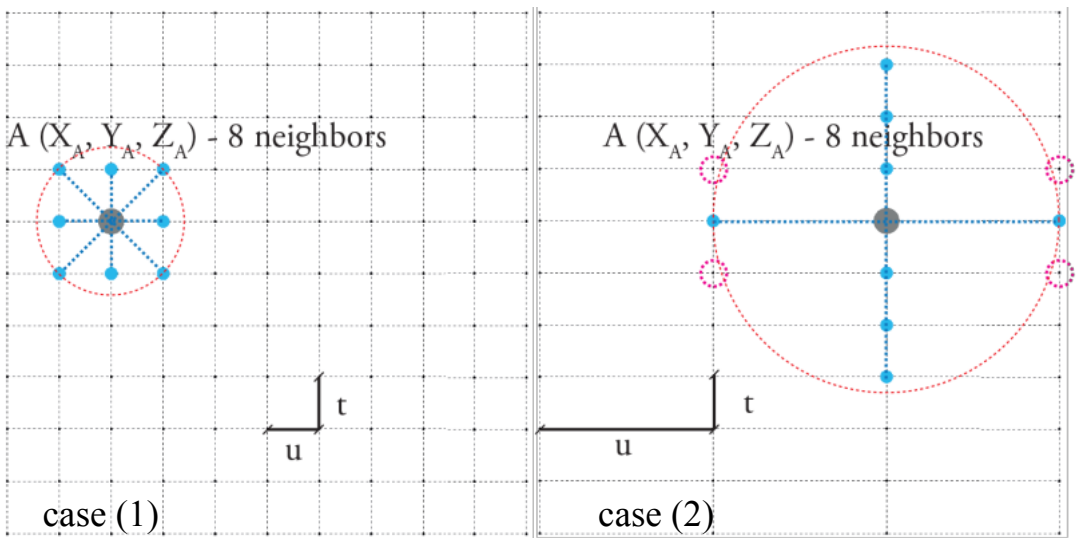

Figure 3. In case (1) the 8 closest points to A can be used to construct graph edges. In case (2), where the ratio between u and t shifts, the same logic fails. This issue is present in both Spiderweb ("graph from points" node) and Tardigrade. An alternative to k-NN and fixed-radius nearest neighbors should be employed.

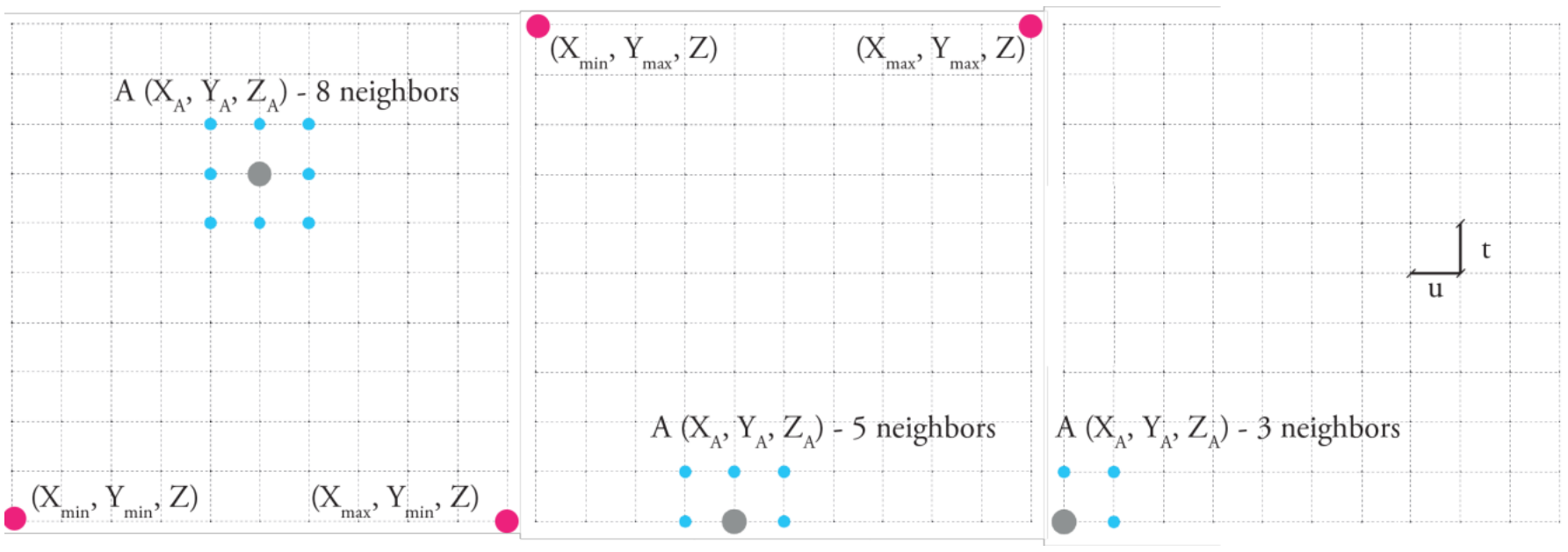

Figure 4. Nodes and neighbor numbers in the graph

3.3 Components

All nodes have a point cloud as input (a collection of 3D points) and a start point. Some components also take in an end point. In terms of output, all nodes retrieve one path from the start point to the endpoint or paths from the start point to all other points in the point cloud.

3.3.1 Breadth-first

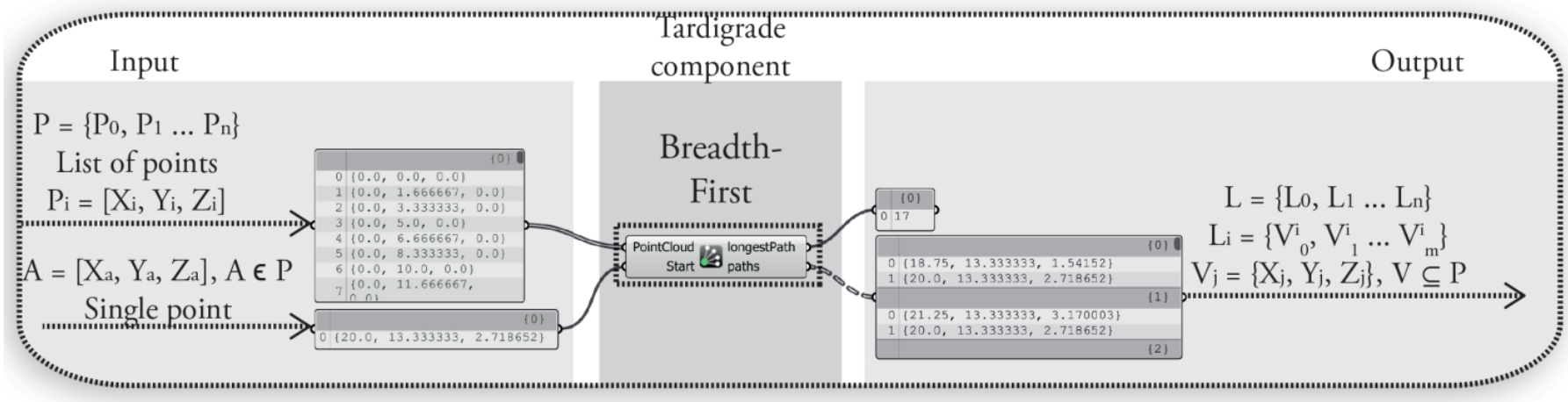

Figure 2. Breadth First search returning all paths

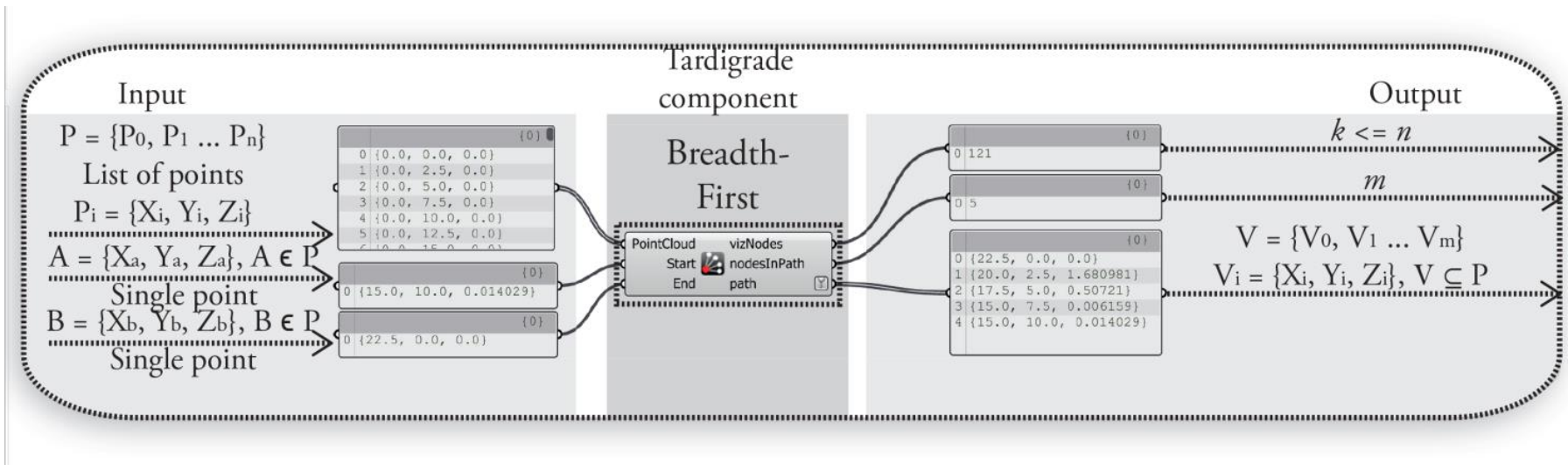

Figure 3. Breadth First search returning one path

The Breadth-First search explores all nodes in a graph. In the implementation employed for Tardigrade its complexity is O(|E|) (E - number of edges in the point cloud). The node in Figure 2 returns all paths from a start point to the other points in the point cloud. The node in Figure 3 returns the shortest path by number of vertices from Start to End.

3.3.2 Greedy Best First

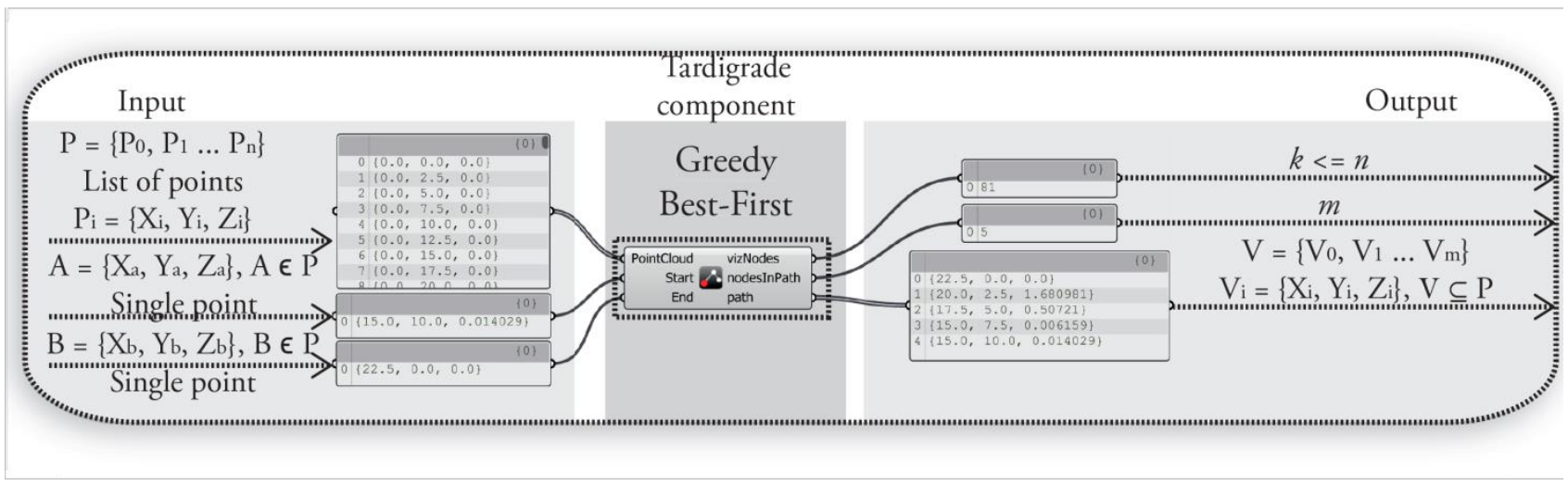

Figure 4. Greedy Best-First search

The Greedy Best-First search operates at a complexity of O(|E|) in the worst case scenario. The algorithm is in most cases faster than Breadth-First. It will stop once it reaches the end-point and internally works with a cost which keeps the direction towards the End point.

3.3.3 Dijkstra

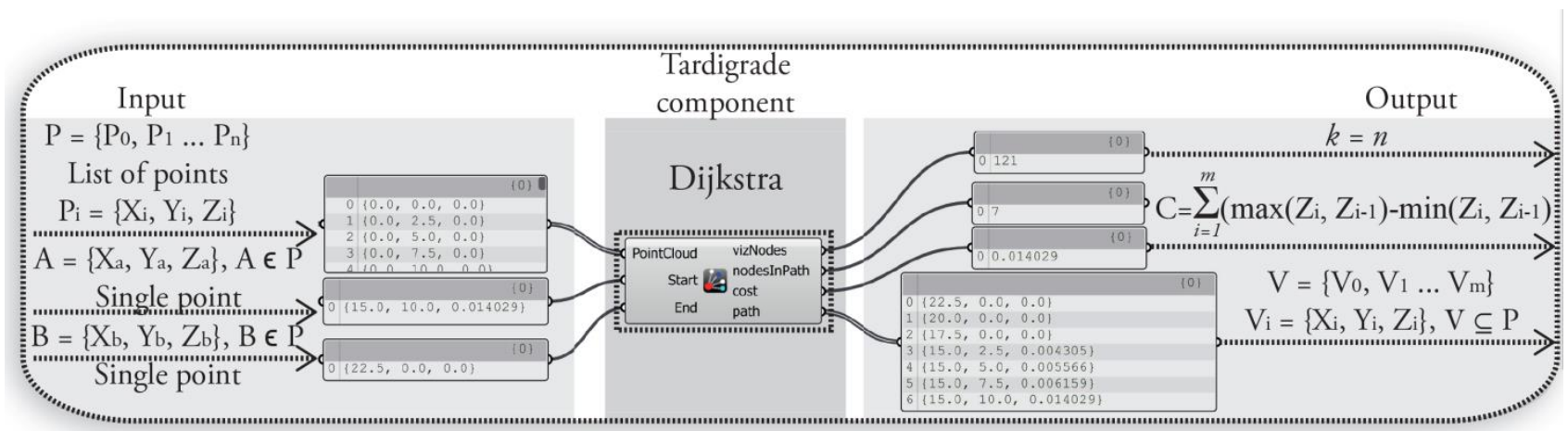

Figure 5. Dijkstra algorithm node

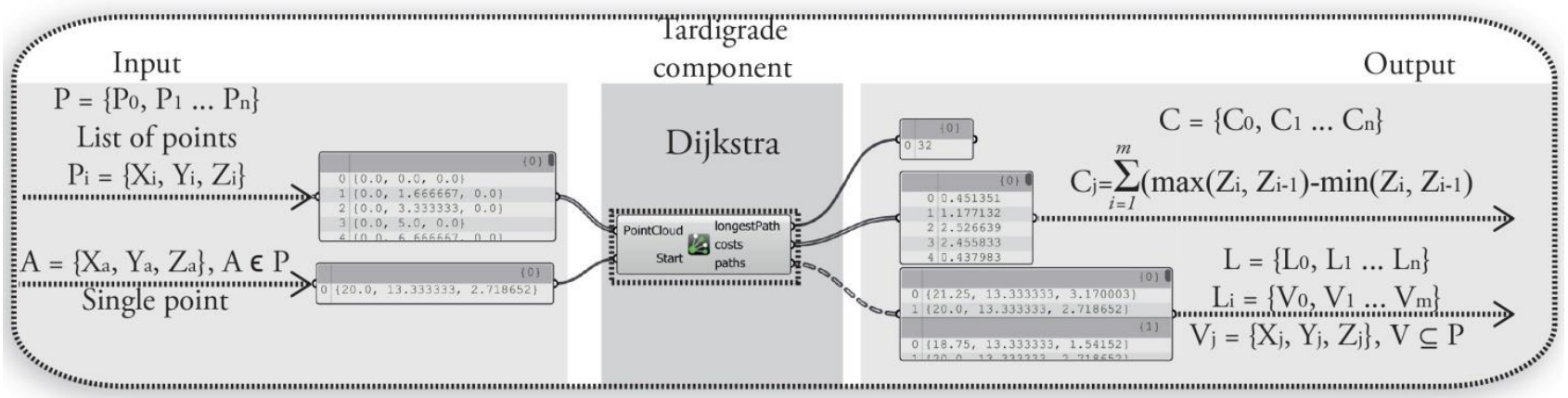

Figure 6. Dijkstra algorithm node

The Dijkstra algorithm finds the shortest cost based path from a start to an end vertex in a graph or retrieves paths from a start to all other nodes in the graph. Dijkstra's complexity is O(|E|+|V|*log|V|). The cost employed in the components developed for the current version of Tardigrade is based on Z coordinate difference. This means the components will retrieve a path from Start to End which is both least steep and shortest (flattest path).

*3.3.4 A**

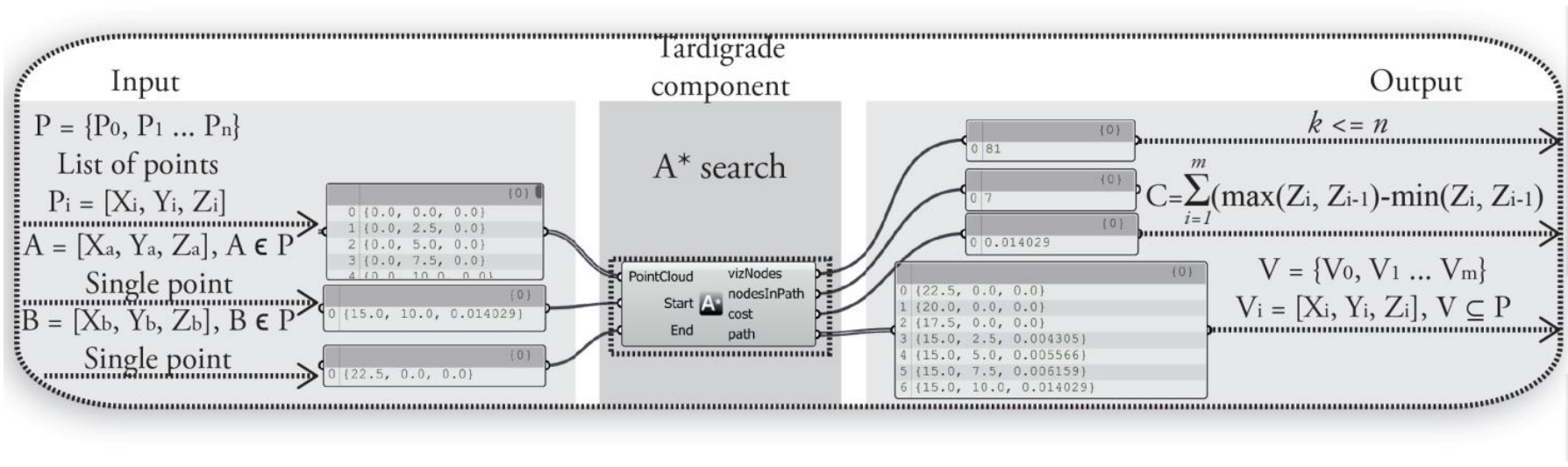

Figure 7. A* search node

The A* graph path finding algorithm is considered one of the most accurate and well-performing graph search algorithms. It is a combination between Dijkstra and Greedy Best-First. Its complexity is O(|E|) in the worst case scenario, performing at a smaller computational time than Dijkstra.

4. Conclusions

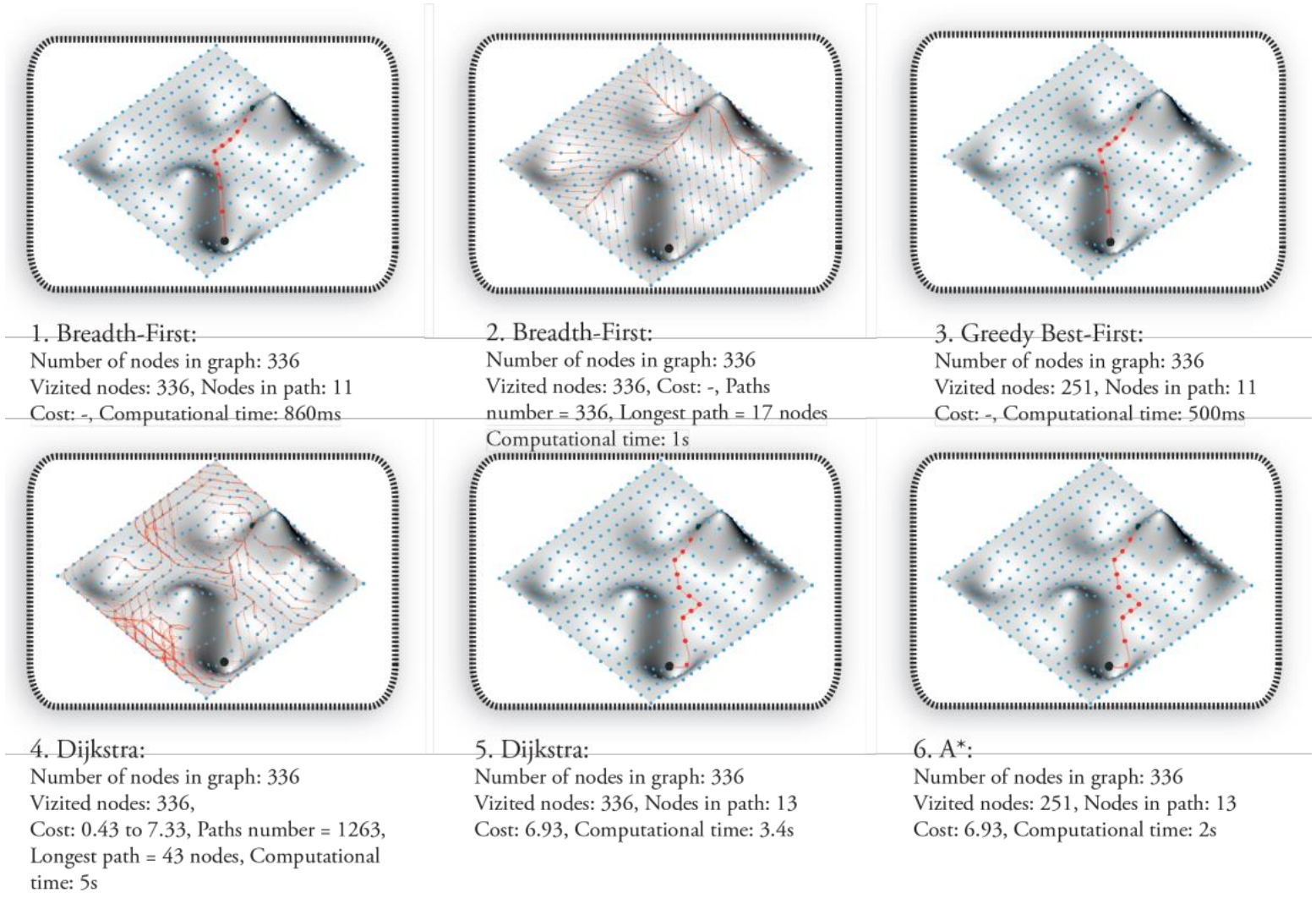

Figure 8. Comparative example of the six components

Figure 8 and Figure 9 present comparative studies of the six components acting upon the same inputs. For high vertex input (above 1000 points), computational time becomes problematic, especially in the case of the nodes employing Dijkstra. It might be an issue with Python being an interpreted language, but faster solutions should be explored. As stated in section 2.2, the edge reconstruction method needs further research and evaluation. The six nodes presented here offer simple methods for path-finding in Grasshopper. The flattest path problem is compressed and a definition of 3-5 nodes, with a cyclomatic complexity of 1. Further research should attach different costs for the path evaluation (i.e. instead of a Z coordinate cost, a solar radiation cost, or a time-based cost or a combination of costs).

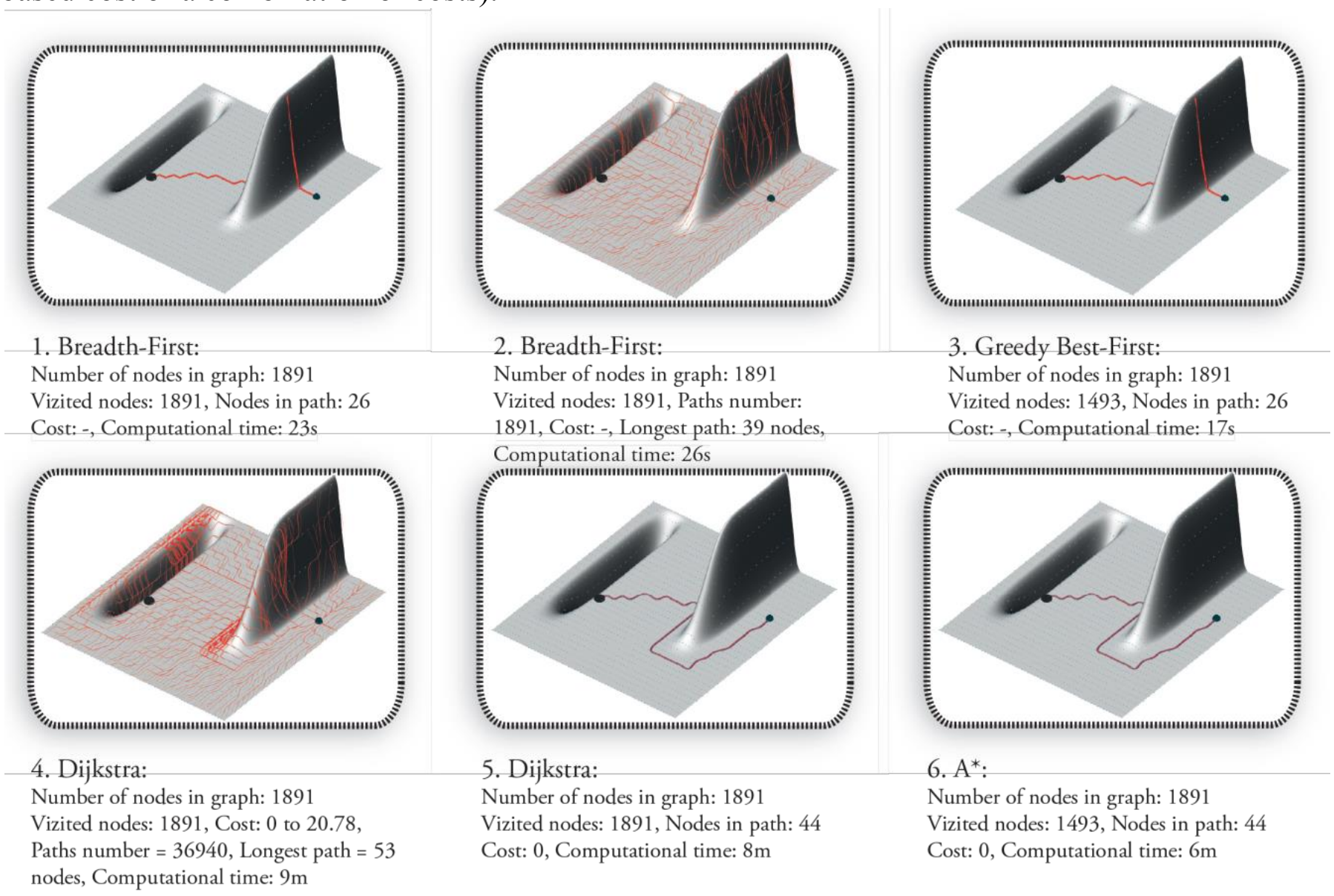

Figure 9. Comparative example of the six components

5. References

[1] M. Teer, "Black Spectacles," Jun 2014. [Online]. Available: http://blackspectacles.com/blog/software-licensure-requirements-to-work-top-50-architecture-firms#.VE_bqxa0T1H. [Accessed 29 October 2014].

[2] A. Nedovizin, N. Gorodetskiy, L. Yng, A. Gimenez and D. McArdle, "Sverchok add-on for Blender," 2011. [Online]. Available: http://wiki.blender.org/index.php/Extensions:2.6/Py/Scripts/Nodes/Sverchok. [Accessed 10 10 2014].

[3] I. Keogh, "Dynamo Bim," 2011. [Online]. Available: http://dynamobim.com/designscript-is-now-dynamo/. [Accessed 10 10 2014].

[4] M. Rhinoceros, "Rhino Python," McNeel Rhinoceros, 2014. [Online]. Available:

http://4.rhino3d.com/5/ironpython/index.html. [Accessed 10 11 2014].

[5] G. Piacentino, "GhPython - Food4Rhino," McNeel Rhinoceros, 2011. [Online]. Available: http://www.food4rhino.com/project/ghpython?ufh. [Accessed 10 10 2014].

[6] N. Miller, "The Proving Ground," CASE, June 2014. [Online]. Available: http://wiki.theprovingground.org/revit-api. [Accessed 29 October 2014].

[7] D. Davis, Modelled on Software Engineering: Flexible Parametric Models in the Practice of Architecture, Melbourne: PhD Dissertation, RMIT University, 2013, p. 7

[8] R. Woodburry, Elements of Parametric Design, Abington: Routledge, 2010, pp. 7-8.

[9] W. Jabi, Parametric Design for Architecture, London: Laurence King, 2013, p. 7.

[10] R. Aish, "Autodesk research - Design Script," Autodesk, 28 June 2012. [Online]. Available: http://www.autodeskresearch.com/publications/designscript. [Accessed 29 October 2014].

[11] P. Dursun, "Space Synax in Architectural Design," in *6th International Space Syntax Symposium*, Istambul, 2007.

[12] T. Varoudis, "Space Syntax," Space Syntax, 2013. [Online]. Available: http://www.spacesyntax.net/software/ucl-depthmapx/. [Accessed 29 October 2014].

[13] R. Schaffranek and M. Vasku, "Space Syntax for Generative Design: on the application of a new tool," in *Proceedings of the Ninth International Space Syntax Symposium*, Seoul, 2013.

[14] A. Turiello, "Food4Rhino," McNeel, 23 November 2012. [Online]. Available: http://www.food4rhino.com/project/shortestwalkgh?etx&ufh. [Accessed 30 October 2014].

[15] A. Patel, "Red Blob Games," Stanford University, 2014. [Online]. Available: http://www.redblobgames.com/. [Accessed 11 11 2014].

[16] K. Hajebi, Y. Abbasi-Yadkori, H. Shahbazi and H. Zhang, "Fast Approximate Nearest-Neighbor Search with k-Nearest Neighbor Graph," in *Proceedings of the Twenty-Second International Joint Conference of Artificial Intelligence*, Barcelona, 2011.

[17] R. Schaffranek, "GBL Digital," TU Wien, 2012. [Online]. Available: http://www.gbl.tuwien.ac.at/_docs/GrasshopperScriptum/GrasshopperScriptum.html?filter=SpiderWeb. [Accessed 30 October 2014].